

Maintenance et évolution d'applications de bases de données

Auteurs

Jean-Marc Hick : Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique
Rue Grandgagnage 21, B-5000 Namur, Belgique
Tél. : +32 81 72 49 85
Fax : +32 81 72 49 67
E-mail : jmh@info.fundp.ac.be

Jean-Luc Hainaut : Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique
Rue Grandgagnage 21, B-5000 Namur, Belgique
Tél. : +32 81 72 49 96
Fax : +32 81 72 49 67
E-mail : jlh@info.fundp.ac.be

Résumé

Les problèmes liés à la maintenance et à l'évolution des systèmes d'information (SI) sont parmi les plus complexes. Cet article propose un cadre de référence méthodologique permettant, d'une part, de décrire les problèmes de l'évolution du composant base de données des SI, et d'autre part, de proposer un ensemble de stratégies de comportement des développeurs face à ces problèmes. L'article décrit ensuite l'environnement de génie logiciel DB-MAIN avec lequel a été construit un prototype d'outil d'aide à l'évolution. Cet outil permet notamment de générer automatiquement les programmes de conversion de la base de données à partir de la trace des opérations de modification du schéma logique de la base de données.

Mots-clés

Evolution d'un SI, conversion de bases de données, évolution, transformation de schéma, AGL.

Abstract

The problems concerning the maintenance and evolution of information systems (IS) are among the most complex ones. This paper presents a methodological framework that makes it possible, on the one hand, to describe the evolution problems of the data component of IS, and on the other hand, to suggest a series of strategies that model the designer behaviour facing those problems. It describes then the DB-MAIN CASE tool environment, with which a prototype of evolution tool has been built. This tool generates automatically the database conversion programs from the modifications history of the database logical schema.

Keywords

IS evolution, database conversion, evolution, schema transformation, CASE tool.

1 Introduction

Les applications de bases de données, au cours de leur cycle de vie, sont amenées à évoluer. Cette évolution provient de changements dans les besoins de l'entreprise ou de l'apparition de nouveaux besoins. Ceux-ci sont de natures diverses : fonctionnels (satisfaction des exigences des utilisateurs), organisationnels (modification du contexte de travail dans lequel s'inscrit l'application) ou techniques (modification des contraintes techniques ou matérielles). Suite à cette évolution des besoins, le concepteur apporte des modifications techniques à l'application.

Confronté au problème de l'évolution, le concepteur est démuné tant du point de vue méthodologique que du point de vue technique. En effet, on ne dispose pas à l'heure actuelle de règles systématiques de traduction des modifications des besoins des utilisateurs en modifications des composants techniques de l'application. On ne dispose même pas de recommandations méthodologiques pour construire des applications stables, robustes et faciles à maintenir. Au niveau des outils, le développeur se voit offrir des ateliers de génie logiciel qui ignorent la plupart du temps les processus de maintenance et d'évolution. En ce qui concerne les bases de données, ces outils permettent de construire un schéma conceptuel, de le transformer de manière automatique en schéma logique et de générer les structures de la base de données. Le code généré doit souvent être remanié de manière significative pour devenir véritablement opérationnel et toute modification des spécifications entraînera la modification du schéma conceptuel, la transformation en un nouveau schéma logique et la production d'un nouveau code. Ce nouveau composant est sans lien formel avec la version précédente. La conversion des données et la modification des programmes est entièrement à la charge du développeur.

L'objectif de cet article est de présenter un modèle abstrait et un environnement d'assistance relatifs à l'évolution des applications de bases de données, considérées comme l'intégration de trois composants : les structures de données, les données et les programmes. L'approche ne fait pas d'hypothèses sur la technologie d'implémentation de l'application¹. Elle est donc applicable à des systèmes développés à l'aide de langages standards de troisième génération tels que COBOL/SQL ou C/SQL.

2 Le modèle de référence

On considère l'analyse des modifications dans le cadre de la modélisation classique qui distingue trois niveaux d'abstraction dans les spécifications (limitées ici à celle des données) : conceptuel, logique et physique (figure 1). Les modifications des besoins se traduisent en modification des spécifications à l'un de ces niveaux. Le problème consiste alors à propager les modifications dans les autres niveaux, en amont ou en aval.

On admet que toute spécification résulte de la transformation d'une autre spécification par une chaîne de transformations élémentaires. C'est ainsi qu'un schéma logique est obtenu par la transformation d'un schéma conceptuel, et qu'un schéma physique est produit par transformation du schéma logique. D'autre part, l'évolution des spécifications à un niveau d'abstraction peut aussi être modélisée comme une transformation. En toute généralité, les transformations élémentaires se classent en trois catégories : T+, celles qui enrichissent les spécifications (ajouter un attribut à un type d'entités), T-, celles qui les réduisent (supprimer un type d'associations) et T=, celles qui les préservent

¹ Cependant, le paradigme objet permet de développer des solutions élégantes grâce au contrôle des versions de schémas et des versions d'instances [Nguyen,89] [Aljadir,95].

(transformer un type d'associations fonctionnel en clé étrangère). A toute transformation t correspond une inverse t' dont l'effet est d'annuler celui de t .

Les transformations $T=$, dites aussi *réversibles*, sont principalement utilisées dans la production de schémas logiques et physiques, tandis que les transformations $T+$ et $T-$ forment la base du processus d'évolution des spécifications.

La trace des opérations de transformation qui ont produit une spécification S_j à partir d'une spécification S_i est appelée *historique* (H_{ij}) du processus de transformation. On utilise la notation fonctionnelle $S_j = H_{ij}(S_i)$. Il est possible de dériver de H_{ij} la fonction inverse H_{ji} telle que $S_i = H_{ji}(S_j)$. H_{ji} peut être obtenue en substituant à chaque opération d'origine son inverse, puis en inversant l'ordre des opérations. Un historique étant de nature procédurale, il ne peut être manipulé aisément que s'il respecte certaines règles de normalisation : en particulier la monotonie (par de retours en arrière) et la linéarité (pas de branchements multiples). Le lecteur intéressé est renvoyé à [Hainaut,96a].

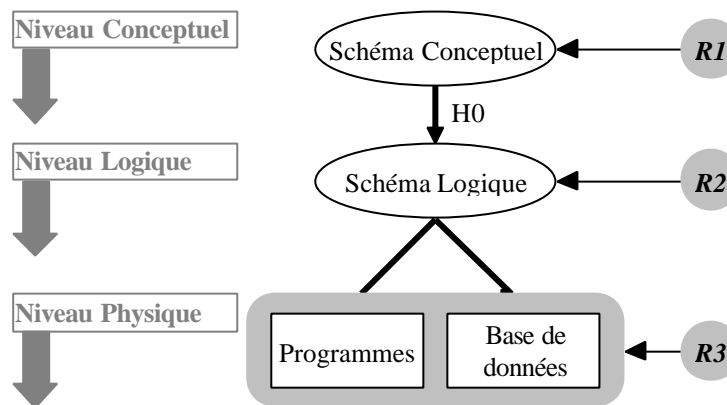


Figure 1 - Modélisation classique définie selon trois niveaux d'abstraction. Le schéma conceptuel répond aux besoins fonctionnels R1, le schéma logique aux besoins R2 et l'application aux besoins techniques R3. H0 est l'historique de la conception logique.

3 Stratégies d'évolution

Les stratégies que nous développerons sont basées sur deux hypothèses. Pour les modifications apportées aux niveaux conceptuel et logique, la version des schémas avant modifications est conservée et le concepteur travaille sur une nouvelle version. Au niveau physique, les modifications sont directement apportées à l'application (modifications des programmes, conversions des structures de données et données). La nouvelle version se substitue complètement à l'ancienne.

Nous définirons trois stratégies de référence qui répondent chacune à des problèmes spécifiques de modification d'une application. Nous limiterons l'étude de ces stratégies au contrôle de la propagation des modifications des spécifications vers les autres niveaux d'abstraction.

3.1 Première stratégie : Modification des spécifications conceptuelles (propagation vers l'aval)

Dans ce premier scénario, les modifications proviennent de changements dans les besoins au niveau conceptuel. Le problème est de propager ces modifications aux couches inférieures (logique et physique). On suppose que les spécifications de tous les niveaux de modélisation existent : le schéma conceptuel SC0, le schéma logique SL0, la base de données D0 (schéma + données) et les

programmes P0 du niveau physique (figure 2a). Les transformations appliquées sur SC0 pour obtenir SL0 (conforme au modèle relationnel) sont enregistrées sous forme de l'historique H0.

Supposons que les besoins auxquels SC0 satisfait évoluent de R1 vers R1'. Le changement est traduit par l'analyste en modifications du schéma SC0 qui devient SC1. Les transformations appliquées pour obtenir SC1 à partir de SC0 sont enregistrées dans l'historique EC1. Une typologie des modifications possibles aux niveaux conceptuel et logique est proposée dans [Roddick,93] et [Hick,97].

La production d'une nouvelle version de l'application consiste d'abord à transformer le schéma SC1 en un schéma logique relationnel SL1 qui soit le plus proche possible de l'ancienne version, mais qui intègre les modifications. Pour ce faire, on rejoue² l'historique H0 sur SC1. Dans le schéma SC1, des structures ont pu être créées, détruites ou modifiées par rapport au schéma SC0. On en déduit donc trois situations de base :

- un objet de SC0 est conservé dans SC1 : il suffit de réappliquer les transformations de H0 sur SC1;
- un objet de SC0 a disparu dans SC1 : les transformations de H0 peuvent être appliquées, mais resteront sans effet;
- un objet absent dans SC0 est introduit dans SC1 : les transformations de H0 resteront sans effet, l'objet devant être traité de manière spécifique.

La réapplication à SC1 de l'historique ancien auquel s'ajoute le traitement des nouveaux objets représente le nouveau processus de conception logique. Il y correspond un nouvel historique H1.

Le schéma logique étant modifié, il reste à convertir la structure de la base de données, les données (D0) et les programmes (P0) qui doivent se conformer aux nouvelles spécifications. La conversion des données (D0) est une tâche déterministe, qui peut donc être automatisée. L'analyse de l'historique EC1 permet de repérer les modifications opérées au niveau conceptuel et l'analyse de H1 nous fournit les informations nécessaires pour en dériver les structures logiques. Sur base de l'analyse de EC1 et H1, on déduit les transformations des structures de données et les transformations des instances de SL0 (c'est-à-dire D0) en instances de SL1 (D1). Ces transformations sont traduites en opérateurs relationnels SQL dans les cas simples, et en programmes dans les situations plus complexes. On obtient ainsi la nouvelle version de la base de données par application d'une chaîne de transformations à l'ancienne.

Modifier les programmes est une tâche beaucoup plus complexe, et pour l'instant non automatisable, sauf dans des situations simples. Nous pensons que le développement d'un outil d'annotation des programmes, qui signale au programmeur les sections de code ou les instructions susceptibles d'être modifiées, ainsi que la nature des modifications à effectuer, constitue un compromis acceptable dans un domaine où il existe peu ou pas de solution. Des techniques d'analyse de programmes telles que les graphes de dépendances et la fragmentation de programmes permettent de localiser avec une bonne précision le code à modifier [Henrard,96].

² *Rejouer un historique* consiste à exécuter sur un schéma les transformations enregistrées dans cet historique.

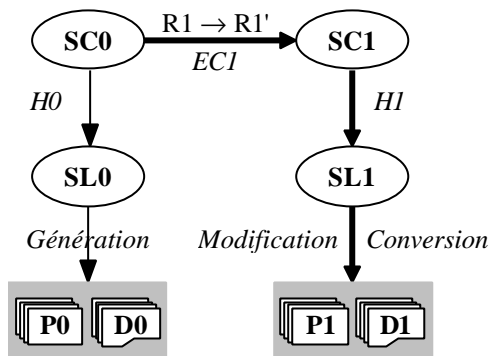


Figure 2a - Propagation vers l'aval des modifications déduites de $R1 \rightarrow R1'$.

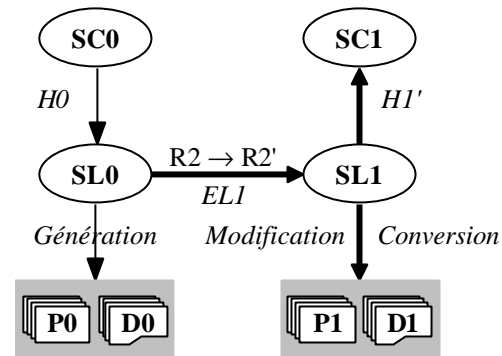


Figure 2b - Propagation vers l'amont et l'aval des modifications déduites de $R2 \rightarrow R2'$.

3.2 Deuxième stratégie : Modification des spécifications logiques (propagation en amont et en aval)

Selon un deuxième scénario, l'analyste apporte des modifications au schéma logique SL0. Ses motivations peuvent être de deux ordres : soit la modification résulte du changement des besoins au niveau logique R2 en R2', soit l'analyste ignore le schéma conceptuel (par exemple parce qu'il n'existe pas) et le changement de besoins conceptuels R1 en R1' est traduit directement en modification du schéma logique (figure 2b). On modifie donc le schéma SL0 pour obtenir le schéma SL1. Les transformations sont enregistrées dans l'historique EL1. Il faut dans ce scénario propager les modifications conduisant à SL1 en aval, vers D1 et P1, mais aussi, autant que possible, en amont, vers SC1.

La propagation en amont n'a de sens que s'il existe un schéma conceptuel SC0. Si ce dernier n'existe pas, il est possible de le retrouver par des techniques de rétro-ingénierie. Le problème de la reconstruction de l'historique H0 à partir de l'historique H0' du processus de rétro-ingénierie est traité dans [Hainaut,96a].

La propagation est basée sur les mêmes principes que ceux qui ont été développés dans la première stratégie. On suppose qu'on dispose de l'historique H0' (résultant de la rétro-ingénierie ou obtenu par inversion de H0 si ce dernier existe). SC1 est obtenu à partir de SL1 en deux phases : application de H0', puis conceptualisation des nouveaux objets de SL1. L'historique de ces deux phases constitue H1', dont l'inversion fournit H1.

La propagation vers le niveau physique a déjà été analysée dans le premier scénario. On peut toutefois préciser que, pour générer les convertisseurs, il suffit d'analyser l'historique EL1 puisqu'on dispose des modifications sur le modèle logique qui est conforme aux structures de données du niveau physique.

3.3 Troisième stratégie : Modification des spécifications physiques

La troisième stratégie consiste à modifier l'application elle-même, et plus particulièrement la structure des données (via des ordres tels que ALTER TABLE ou DROP TABLE par exemple), et, partiellement, les données. Ce type de comportement s'observe en général lorsqu'aucune documentation n'est disponible. Les modifications s'effectuant sur le code physique, il est nécessaire de les reporter au moins dans le schéma logique, puis idéalement aussi dans le schéma conceptuel. En effet, la répercussion des modifications physiques vers les données et les programmes est déclenchée par l'évolution du schéma logique. Cette situation est traitée en trois étapes.

1. Reconstruction du schéma logique (et conceptuel) de la base de données.

2. Identification des opérations de modification physique.
3. Propagation des modifications vers les données et les traitements.

L'étape 1 relève de la rétro-ingénierie, et plus particulièrement de sa première phase, l'*extraction des structures de données*, dont le but est la recherche des structures et contraintes explicites et implicites grâce à l'analyse de diverses sources telles que les fragments éventuels de documentation, le code DDL, les programmes, les données, ainsi que les rapports, écrans et formulaires [Hainaut,93].

L'étape 2 consiste en une analyse différentielle des structures physiques, qui est automatisable, du moins lorsque les objets physiques n'ont pas été renommés. Cette analyse implique la conservation (ou la reconstruction) des scripts de création des structures avant et après modifications, la comparaison des deux versions permettant d'identifier les modifications physiques et d'en déduire les modifications au niveau logique. Le renommage des objets complique cette analyse par la rupture de la chaîne de désignation de ces objets. On suggère alors de regrouper ces opérations en un bloc distinct des autres modifications.

L'étape 3 est similaire au processus de la deuxième stratégie.

4 Implémentation

Cette recherche est réalisée dans le cadre du projet DB-MAIN consacré aux problèmes de maintenance et d'évolution des applications de bases de données. L'une des activités principales du projet est la conception d'un AGL en ingénierie et rétro-ingénierie de bases de données qui tente d'apporter des réponses aux lacunes des outils existants [Hainaut,96b]. Les outils supportant les stratégies développées au point 3 sont implémentés dans l'atelier. Nous rappelons d'abord les composants et les fonctions qui vont aider à l'évolution d'applications. Ensuite, nous décrivons le cheminement à suivre dans l'atelier pour propager les modifications.

L'atelier met à disposition des fonctions et des composants d'usage très général, et qui permettent en particulier le développement de processeurs assurant la gestion de la maintenance et de l'évolution :

- un modèle générique de représentation de schémas basé sur un modèle Entité/Objet-Association étendu qui permet de représenter les structures de données à tous les niveaux d'abstraction;
- une approche transformationnelle concrétisée par une trentaine de transformations applicables aussi bien en conception qu'en rétro-ingénierie;
- l'enregistrement dans un historique des opérations effectuées sur un schéma ainsi que des fonctions de gestion de cet historique (sauvegarder, rejouer, inverser un historique, ...);
- un langage de quatrième génération (Voyager 2) qui permet au concepteur de personnaliser l'atelier et de concevoir ces propres fonctions. C'est dans ce langage qu'ont été implémentés les convertisseurs de données.

Au départ, l'existant est composé du schéma conceptuel SC0, du schéma logique SL0, de l'historique H0 de la traduction de SC0 en SL0, de la base de données D0 (scripts de création des structures de données relationnelles + données) et des programmes P0 (fichiers contenant les programmes). Si les composants SC0, SL0 ou H0 sont manquants, ils sont d'abord reconstitués par des techniques de rétro-ingénierie.

Dans le cas de la première stratégie, le concepteur opère comme suit.

- *Production de la nouvelle version du schéma conceptuel* : le concepteur demande une copie de SC0, nommée SC1, et modifie SC1 en déclenchant l'enregistrement des transformations dans l'historique EC1.
- *Production de la nouvelle version du schéma logique (phase 1)* : le concepteur demande une copie de SC1 (nommée SL1) sur laquelle il rejoue l'historique H0.
- *Production de la nouvelle version du schéma logique (phase 2)* : les composants nouveaux éventuellement ajoutés à SC0 sont alors transformés spécifiquement selon le plan de transformation en vigueur dans l'entreprise. L'historique de ces transformations s'ajoute à H0 pour produire H1.
- *Conversion de la base de données* : le concepteur exécute le convertisseur sur base de EC1 et H1. Il obtient ainsi un script de conversion des structures de données et des données. Il exécute ce script.
- *Conversion des programmes* : le concepteur exécute un analyseur de programme qui repère les composants de la base de données qui ont été modifiés. L'analyseur produit un rapport des modifications à effectuer.

La deuxième stratégie fait appel aux mêmes techniques que la première, sauf en ce qui concerne l'élaboration des historiques H0' et H1', qui s'obtiennent grâce à un inverseur d'historique.

Quant à la troisième stratégie, elle est basée en outre sur un analyseur différentiel de schémas physiques.

Les processeurs nécessaires à la gestion de l'évolution existent dans la plate-forme de base de l'atelier, à l'exception des générateurs de scripts, des générateurs de rapports de modification des programmes et de l'analyseur différentiel, qui sont développés en Voyager 2.

5 Conclusions

Le problème de l'évolution d'un système d'information inclut celui des données qui en constituent le noyau. L'évolution des besoins se traduit techniquement par la modification des spécifications d'un des niveaux d'abstraction de ces données. La difficulté réside dans la propagation de cette modification vers les autres niveaux, et en particulier celui des composants opérationnels : schéma physique, données et programmes.

Les concepts de l'approche DB-MAIN, dédiée à l'ingénierie des données, forment une base formelle favorable à la compréhension et à la résolution du problème de l'évolution : modélisation transformationnelle des processus, représentation uniforme rigoureuse des spécifications aux différents niveaux d'abstraction et selon différents paradigmes, traçabilité des processus. Si les documents de la conception d'un système sont encore disponibles, ou à défaut, peuvent être reconstitués par rétro-ingénierie, alors le contrôle de l'évolution devient un processus formellement défini, et donc déterministe, pour ce qui concerne la base de données. En revanche, la modification des programmes reste un problème ouvert dans le cas général. Il est cependant possible d'aider le développeur à modifier le code de ces programmes par le repérage des sections où des instances des types d'objets modifiés sont traitées.

Nous disposons actuellement d'un prototype de contrôle de l'évolution de base de données relationnelle selon la deuxième stratégie mentionnée dans cet article. Ce prototype a été développé en Voyager 2 sur la plate-forme générique DB-MAIN. Il est actuellement possible de générer de manière automatique les programmes de conversion d'une base de données correspondant à une séquence de modifications quelconques des types T- ou T+. Les règles relatives aux modifications

conceptuelles sont définies, mais restent à implémenter. Il en est de même de la modification des traitements.

6 Références

- [Aljadir,95] L. Al-Jadir, T. Estier, G. Falquet, M. Léonard, *Evolution features of the F2 OODBMS*, in proc. of the 4th International Conf. on Database Systems for Advanced Applications, Singapore, World Scientific Publishing, Avril 1995.
- [Hainaut,93] J.-L. Hainaut, M. Chandelon, C. Tonneau, M. Joris, *Contribution to a theory of database reverse engineering*, in proc. of the IEEE Working Conf. on Reverse Engineering, Baltimore, IEEE Computer Press, May 1993.
- [Hainaut,96a] J.-L. Hainaut, J. Henrard, J.-M. Hick, D. Roland, V. Englebert, *Database Design Recovery*, in proc. of the 8th Conf. on Advanced Information Systems Engineering, Springer-Verlag, 1996.
- [Hainaut,96b] J.-L. Hainaut, V. Englebert, J. Henrard, J.-M. Hick, D. Roland, *Database Reverse Engineering: from requirements to CARE tools*, in Journal of Automated Software Engineering 3(2), Kluwer Academic Press, 1996.
- [Henrard,96] J. Henrard, J.-M. Hick, D. Roland, V. Englebert, J.-L. Hainaut, *Technique d'analyse de programmes pour la rétro-ingénierie de bases de données*, dans les actes du 14ème congrès INFORSID, Bordeaux, Juin 1996.
- [Hick,97] J.-M. Hick, *Typologie des modifications d'une application de base de données relationnelles*, rapport de recherche DB-MAIN, Institut d'informatique, FUNDP, Mai 1997.
- [Nguyen,89] G.T. Nguyen, D. Rieu, *Schema evolution in object-oriented database systems*, in Data & Knowledge Engineering (4), Elsevier Science Publishers, 1989.
- [Roddick,93] J.F. Roddick, N.G. Crashe, T.J. Richards, *A Taxonomy for Schema Versioning Based on the Relational and Entity Relationship Models*, in proc. of 12th International Conf. on the Entity-Relationship Approach, Arlington, LNCS, 1993.